Lecture 6
Greedy Algorithms
- Minimum Spanning Tree
  - Kruskal’s, Prim’s
  - Reverse-Delete
- Clustering
- Huffman Codes

Sofya Raskhodnikova
Review Question

• Is Dijkstra’s algorithm correct with negative edge weights?
Minimum Spanning Tree
**Cut property.** Let S be a subset of nodes. Let e be the min weight edge with exactly one endpoint in S. Then the MST contains e.

**Cycle property.** Let C be a cycle, and let f be the max weight edge in C. Then the MST does not contain f.
Review Questions

Let $G$ be a connected undirected graph with distinct edge weights. Answer true or false:

- Let $e$ be the cheapest edge in $G$. The MST of $G$ contains $e$.

- Let $e$ be the most expensive edge in $G$. The MST of $G$ does not contain $e$. 

Let $G$ be a connected undirected graph with distinct edge weights. Answer true or false:

- Let $e$ be the cheapest edge in $G$. The MST of $G$ contains $e$.
  
  (Answer: True, by the Cut Property)

- Let $e$ be the most expensive edge in $G$. The MST of $G$ does not contain $e$.
  
  (Answer: False. Counterexample: if $G$ is a tree, all its edges are in the MST.)
Greedy Algorithms for MST

• **Kruskal's:** Start with $T = \emptyset$. Consider edges in ascending order of weights. Insert edge $e$ in $T$ unless doing so would create a cycle.

• **Reverse-Delete:** Start with $T = E$. Consider edges in descending order of weights. Delete edge $e$ from $T$ unless doing so would disconnect $T$.

• **Prim's:** Start with some root node $s$. Grow a tree $T$ from $s$ outward. At each step, add to $T$ the cheapest edge $e$ with exactly one endpoint in $T$. 
Prim's Algorithm: Correctness

• Prim's algorithm. [Jarník 1930, Prim 1959]
  – Apply cut property to T.
  – When edge weights are distinct, every edge that is added must be in the MST.
  – Thus, Prim’s algorithm outputs the MST.
Correctness of Kruskal

- [Kruskal, 1956]: Consider edges in ascending order of weight.
  - **Case 1**: If adding $e$ to $T$ creates a cycle, discard $e$ according to cycle property.
  - **Case 2**: Otherwise, insert $e = (u, v)$ into $T$ according to cut property where $S = \text{set of nodes in } u's\ connected\ component$.  

S. Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne

L6.9
Lexicographic Tiebreaking

- To remove the assumption that all edge costs are distinct: perturb all edge costs by tiny amounts to break any ties.

- **Impact.** Kruskal and Prim only interact with costs via pairwise comparisons. If perturbations are sufficiently small, MST with perturbed costs is MST with original costs.

- **Implementation.** Can handle arbitrarily small perturbations implicitly by breaking ties lexicographically, according to index.

```java
boolean less(i, j) {
    if (cost(e_i) < cost(e_j)) return true
    else if (cost(e_i) > cost(e_j)) return false
    else if (i < j) return true
    else return false
}
```

e.g., if all edge costs are integers, perturbing cost of edge $e_i$ by $i / n^2$
Implementing MST algorithms

- **Prim**: similar to Dijkstra

- **Kruskal**:
  - Requires efficient data structure to keep track of “islands”: Union-Find data structure
Implementation of Prim($G, w$)

**IDEA:** Maintain $V - S$ as a priority queue $Q$ (as in Dijkstra). Key each vertex in $Q$ with the weight of the least-weight edge connecting it to a vertex in $S$.

$Q \leftarrow V$

$key[v] \leftarrow \infty$ for all $v \in V$

$key[s] \leftarrow 0$ for some arbitrary $s \in V$

while $Q \neq \emptyset$

\[
\text{do } u \leftarrow \text{EXTRACT-MIN}(Q)
\]

\[
\text{for each } v \in \text{Adjacency-list}[u]
\]

\[
\text{do if } v \in Q \text{ and } w(u, v) < key[v]
\]

\[
\text{then } key[v] \leftarrow w(u, v) \quad \triangleright \quad \text{DECREASE-KEY}
\]

\[
\pi[v] \leftarrow u
\]

At the end, $\{(v, \pi[v])\}$ forms the MST.
Analysis of Prim

\( Q \leftarrow V \)
\( key[v] \leftarrow \infty \) for all \( v \in V \)
\( key[s] \leftarrow 0 \) for some arbitrary \( s \in V \)

while \( Q \neq \emptyset \)
do \( u \leftarrow \text{EXTRACT-MIN}(Q) \)
for each \( v \in \text{Adj}[u] \)
do if \( v \in Q \) and \( w(u, v) < key[v] \)
then \( key[v] \leftarrow w(u, v) \)
\( \pi[v] \leftarrow u \)

\( \Theta(n) \) total
\( n \) times
\( \Theta(m) \) implicit DECREASE-KEY’s.

Time: as in Dijkstra
Greedy Algorithms for MST

• **Kruskal's**: Start with \( T = \emptyset \). Consider edges in ascending order of weights. Insert edge \( e \) in \( T \) unless doing so would create a cycle.

• **Reverse-Delete**: Start with \( T = E \). Consider edges in descending order of weights. Delete edge \( e \) from \( T \) unless doing so would disconnect \( T \).

• **Prim's**: Start with some root node \( s \). Grow a tree \( T \) from \( s \) outward. At each step, add to \( T \) the cheapest edge \( e \) with exactly one endpoint in \( T \).
Implementation of Kruskal

- Use the **Union-Find** data structure.
  - Build set $T$ of edges in the MST.
  - Maintain a set for each connected component.

```plaintext
Kruskal(G, w) {
    Sort edges weights so that $w_1 \leq w_2 \leq \ldots \leq w_m$.
    $T \leftarrow \emptyset$
    foreach ($u \in V$) make a set containing singleton $u$
    foreach edge $(u,v)$
        //go through edges in sorted order
        if ($u$ and $v$ are in different sets) {
            $T \leftarrow T \cup \{e_i\}$
            merge the sets containing $u$ and $v$
        }  \text{merge two components}
    return $T$
}
```

S.Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne
The Union-Find Data Structure

- **Operations:**
  
  **MAKE-UNION-FIND(S):** creates the data structure; puts all elements in S into separate sets.
  
  \[ O(n) \text{ time where } n=|S| \]

  **FIND(u):** returns the representative of the set containing u.
  
  \[ O(\log n) \text{ time} \]

  **UNION(A,B):** merge sets A,B into a single set.
  
  \[ O(1) \text{ time} \]
Forest Representation

• Each element is a node.
• Each tree represents one set (store its size).
• The root is the representative.
• MAKE-UNION-FIND: create roots
  – $O(1)$ time per element
• UNION(A,B): point the root of the smaller tree to the root of the larger tree
  – $O(1)$ time
FIND operation

• FIND(x): follow the links to the root.

Theorem. FIND takes $O(\log n)$ time.

Proof: Time to evaluate FIND(x)

= number of predecessors of x
= number of times x changes representatives.

• Every time x changes representatives, the size of its set at least doubles. It can happen $\leq \log_2 n$ times. •
An Improvement to FIND

• **Path Compression:** update every pointer on the way to the root.

![Diagram showing path compression]

• **Theorem.** $n$ FIND operations take $O(n \alpha(n))$ time, where $\alpha(n)$ is inverse Ackerman function.
Implementation of Kruskal

• Build set T of edges in the MST.
• Maintain a set for each connected component.

Kruskal(G, w) {
    Sort edges weights so that $w_1 \leq w_2 \leq \ldots \leq w_m$.
    T ← φ
    MAKE-UNION-FIND(V)
    foreach edge (u,v) //go through edges in sorted order
        if (FIND(u) ≠ FIND(v)) {
            T ← T ∪ {e_i}
            UNION(FIND(u), FIND(v))
        }
    return T
}

• Sorting: $O(m \log m) = O(m \log (n^2)) = O(m \log n)$
• Union-Find operations: $O(m \log n)$

$O(m \log n)$ time

S.Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne
MST Algorithms in 2011

• **Deterministic** comparison-based algorithms.
  – $O(m \log n)$ [Jarník, Prim, Dijkstra, Kruskal, Boruvka]
  – $O(m \alpha (m, n))$. [Chazelle 2000]

• **Holy grail:** $O(m)$.

• **Related.**
  – $O(m)$ **randomized.** [Karger-Klein-Tarjan 1995]
  – $O(m)$ **verification.** [Dixon-Rauch-Tarjan 1992]
Clustering
Clustering

Given a set $U$ of $n$ objects (e.g., photos, documents, microorganisms) labeled $p_1, \ldots, p_n$, classify them into coherent groups.

Outbreak of cholera deaths in London in 1850s.
Reference: Nina Mishra, University of Virginia
Clustering

**Distance function:** for each pair of objects specifies how “close” they are.

**Fundamental problem:** Divide into clusters so that objects in different clusters are far apart.

- Routing in mobile ad hoc networks.
- Identify patterns in gene expression.
- Document categorization for web search.
- Similarity searching in medical image databases
- Skycat: cluster $10^9$ sky objects into stars, quasars, galaxies.
Clustering of Maximum Spacing

• **k-clustering:** Divide objects into k non-empty groups.

• Distance function satisfies:
  - \( d(p_i, p_j) = 0 \) iff \( p_i = p_j \) (identity of indiscernibles)
  - \( d(p_i, p_j) \geq 0 \) (nonnegativity)
  - \( d(p_i, p_j) = d(p_j, p_i) \) (symmetry)

• **Spacing:** Minimum distance between any pair of points in different clusters.

• **Goal:** Given an integer k, find a k-clustering of maximum spacing.
Greedy Clustering Algorithm

• Single-link k-clustering algorithm.
  – Form a graph on the vertex set $U$, corresponding to $n$ clusters.
  – Find the closest pair of objects such that each object is in a different cluster, and add an edge between them.
  – Repeat $n-k$ times until there are exactly $k$ clusters.

• Key observation. This procedure is precisely Kruskal's algorithm (except we stop when there are $k$ connected components).

• Remark. Equivalent to finding an MST and deleting the $k-1$ most expensive edges.
Analysis of Greedy Clustering

**Theorem.** Let $C^*$ be the clustering $C^*_{1}, \ldots, C^*_{k}$ formed by deleting $k-1$ most expensive edges of a MST. $C^*$ is a $k$-clustering of max spacing.

**Proof:** Let $C$ be some other clustering $C_1, \ldots, C_k$.

- The spacing of $C^*$ is the length $d^*$ of $(k-1)^{st}$ most expensive edge.
- Let $p_i, p_j$ be in the same cluster in $C^*$, say $C^*_{r}$, but different clusters in $C$, say $C_s$ and $C_t$.
- Some edge $(p, q)$ on $p_i$-$p_j$ path in $C^*_{r}$ spans two different clusters in $C$.
- All edges on $p_i$-$p_j$ path have length $\leq d^*$ since Kruskal chose them.
- Spacing of $C$ is $\leq d^*$ since $p$ and $q$ are in different clusters. $\blacksquare$
Huffman codes
Prefix-free codes

• **Binary code** maps characters in an alphabet (say \{A,\ldots,Z\}) to binary strings

• **Prefix-free code**: no codeword is a prefix of any other
  – ASCII: prefix-free (all symbols have the same length)
  – Not prefix-free:
    • a → 0
    • b → 1
    • c → 00
    • d → 01
    • ...

• Why is prefix-free good?
A prefix-free code for a few letters

A tree for "this is an example of a huffman tree"

- e.g. e → 00, p → 10011


S. Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne
How good is a prefix-free code?

• Given a text, let $f[i] = \# \text{ occurrences of letter } i$
• Total number of symbols needed

$$\sum_{i} f[i] \cdot \text{depth}(i)$$

• How do we pick the best prefix-free code?
Huffman’s Algorithm (1952)

- Given individual letter frequencies $f[1, .., n]$:
  - Find the two least frequent letters $i,j$
  - Merge them into symbol with frequency $f[i]+f[j]$
  - Repeat

- e.g.
  - $a$: 6
  - $b$: 6
  - $c$: 4
  - $d$: 3
  - $e$: 2

**Theorem:** Huffman algorithm finds an optimal prefix-free code.
Warming up

- **Lemma 0**: Every optimal prefix-free code corresponds to a **full** binary tree.
  - (Full = every node has 0 or 2 children)

- **Lemma 1**: Let x and y be two least frequent characters. There is an optimal code in which x and y are siblings.
  - Proof using an exchange argument (on board).

9/7/2011

S. Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne
Huffman codes are optimal

Proof by induction

• Base case: two symbols; only one full tree.

• Induction step:
  – Suppose $f[1], f[2]$ are smallest in $f[1,\ldots,n]$
  – $T$ is an optimal code for $\{1,\ldots,n\}$
  – Lemma 1 $\Rightarrow$ can choose $T$ where 1,2 are siblings.
  – $T'$ = code obtained by merging 1,2 into $n+1$
Cost of $T$ in terms of $T'$:

$$cost(T) = \sum_{i=1}^{n} f[i] \cdot depth(i)$$

$$= \sum_{i=3}^{n+1} f[i] \cdot depth(i) + f[1] \cdot depth(1) + f[2] \cdot depth(2) - f[n+1] \cdot depth(n+1)$$

$$= cost(T') + f[1] \cdot depth(1) + f[2] \cdot depth(2) - f[n+1] \cdot depth(n+1)$$

$$= cost(T') + (f[1] + f[2]) \cdot depth(T) - f[n+1] \cdot (depth(T) - 1)$$

$$= cost(T') + f[1] + f[2]$$

- Minimizing $cost(T)$ is the same as minimizing $cost(T')$.
- By induction hypothesis $T'$ is optimal.
- So, $T$ is optimal, too.
• See Jeff Erickson’s lecture notes on greedy algorithms:
  – http://theory.cs.uiuc.edu/~jeffe/teaching/algorithms/
  – efficient implementation using min-heap
Data Compression for real?

• Generally, we don’t use letter-by-letter encoding
• Instead, find frequently repeated substrings
  – Lempel-Ziv algorithm extremely common
  – also has deep connections to entropy